DESIGN AND IMPLEMENTATION OF A CONTENT AWARE IMAGE PROCESSING MODULE ON FPGA

A Dissertation
Presented to
The Academic Faculty

By

Burhan Ahmad Mudassar

In Partial Fulfillment
Of the Requirements for the Degree
Masters of Science in
School of Electrical and Computer Engineering

Georgia Institute of Technology
May 2015

DESIGN AND IMPLEMENTATION OF A CONTENT AWARE IMAGE PROCESSING MODULE ON FPGA

Approved By

Dr. Saibal Mukhopadhyay, Advisor
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Sudhakar Yalamanchili
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Arijit Raychowdhury
School of Electrical and Computer Engineering
Georgia Institute of Technology

Date Approved: April 22, 2015

To my parents for their love and support

ACKNOWLEDGEMENTS

I would like to thank my advisor Dr. Saibal Mukhopadhyay for his constant support throughout my research work here at Georgia Tech. I would also like to thank all the GREEN Lab Members especially Mr. Jong Hwan Ko and Dr. Denny Lie for their guidance and help. In the end, I would like to thank my parents without whom none of this would have been possible.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS AND KEYWORDS

ROI:	Region of Interest
ED:	Edge Detection
FD:	Frame Differencing
RTL:	Register Transfer Level
FPGA:	Field Programmable Gate Array
BS:	Background Subtraction
JPEG:	Joint Photographic Experts Group
QF:	Quality Factor (JPEG)
FIFO:	First In First Out
YCbCr:	Luminance, Chrominance Blue, Chrominance Red Color Map
nRF:	nRF24L01+ transceiver module
FF:	Flip Flop
LUT:	Look up Table

SUMMARY

In this thesis, we tackle the problem of designing and implementing a wireless video sensor network for a surveillance application. The goal was to design a low power content aware system that is able to take an image from an image sensor, determine blocks in the image that contain important information and encode those block for transmission thus reducing the overall transmission effort. At the same time, the encoder and the preprocessor must not consume so much computation power that the utility of this system is lost.

We have implemented such a system which uses a combination of Edge Detection and Frame Differencing to determine useful information within an image. A JPEG encoder then encodes the important blocks for transmission. An implementation on a FPGA is presented in this work. This work demonstrates that preprocessing gives us a 48.6 % reduction in power for a single frame while maintaining a delivery ratio of above 85 % for the given set of test frames.

CHAPTER 1

INTRODUCTION

An analysis of wireless video transmission systems reveals that the bulk of power draw comes from the transmission of thousands of pixels that constitute the image. Wireless transmission is an expensive process both in terms of power and transmission bandwidth. The goal of this project is to explore low power pre-processing methods that can reduce the amount of content that needs to be transmitted thus saving transmission power and much needed transmission bandwidth.

The next question that arises is how to determine which parts of the image are of importance to us. The answer to that is relatively simple i.e. only moving objects and edges are of interest to us. Transmitting a static background or static objects pose no value to us and will only consume valuable bandwidth. There are many algorithms and techniques that can be used to perform motion estimation and extract moving objects from an image frame.

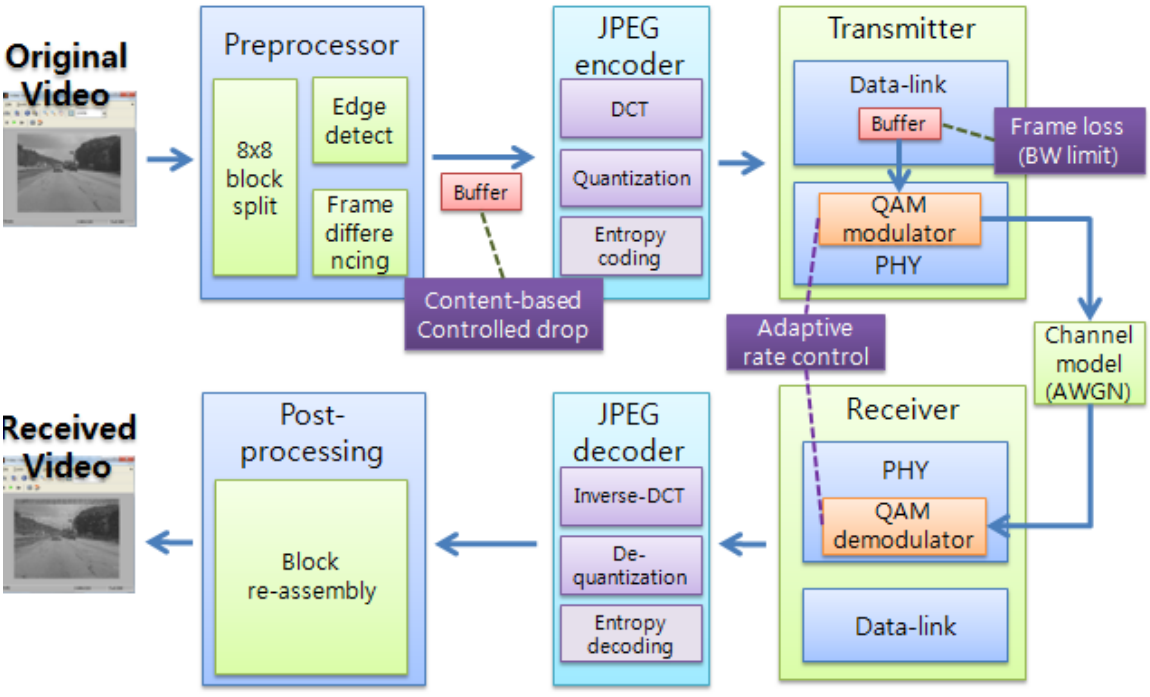


Figure 1 System Framework

Some of the techniques used in this implementation are edge detection, frame differencing or a combination of both. We will analyze results from both techniques to see which suits our design better. A functional block diagram of such a system is presented in figure 1.

After preprocessing the image is encoded to further reduce the total amount of data that needs to be transmitted. There exist a lot of commercial encoding standards that give a satisfactory reduction in the amount of data. Furthermore, we will also be implementing an architecture that performs all these function while consuming as little power as possible. A balance needs to be achieved between computation power and transmission power so that an optimum power consumption value can be attained keeping in mind the strict energy requirements of wireless sensor networks.

1.1 Goal of Thesis

The goal of this thesis is as follows

- Design an architecture in RTL that performs the following functions
    - Preprocessing of the image (determine ROI)
    - Encoding of important blocks only
    - Quality factor for encoding and threshold values are reconfigurable based on channel conditions
- Optimize the architecture to consume as little power as possible while meeting the target frame rate
- Synthesize and implement on a FPGA and verify functionality

1.2 Literature review

Sobel edge detection was first presented for a hardware chip in [1] designed primarily for a military application. In [2] the authors concentrate more on the development of a SOC with a custom image sensor and using FD as the motion detection algorithm with energy harvesting.

In [3] the authors are using FD within the image sensor pixel and wake-up feature extraction. However it only does full frame sensing when an object of interest is detected using feature extraction which has shown a 94.5% success rate for human detection. In [4] the authors are using FD for motion detection and are storing whole image within a frame buffer. [5] used FD to do motion estimation and only transmits frames with a high change in pixel values.

In [6] the authors present a prioritization technique for classifying blocks within an image as “important” or not “important” and only transmitting those blocks which are important. Multiple measures are used to determine the importance of a block e.g. edge measure, entropy measure or a combination of both.

In all these works, FD or background subtraction is the key element used to do motion estimation. However, FD increases computational and storage complexity. A simulation framework demonstrating combination of ED and FD is presented in [7] and which is implemented in this work. ED is used to get a one bit per pixel edge map which is then subjected to FD.

1.3 Image Preprocessing Algorithm

Image preprocessing is done to determine ROI within the frame. Once the ROI are determined, they are chosen for encoding and transmission. Preprocessing is done based on the criteria of moving objects as the background, once transmitted, is of little use to us. A couple of techniques are examined for preprocessing including edge detection, frame differencing and a combination of both.

1.3.1 Edge Detection

Edge Detection methods are a set of tools that can be used to find the amount of change that occurs between pixels within an image. It is an excellent method for extracting the boundaries of objects. For example, edge detection can be used to extract facial features as can be seen in figure 2. All edge detection algorithms work on the principle of differentiation or gradients to detect changes in the brightness levels of a picture.

Figure 2 Edge Detection Example [8]

Some well-known kernels that are used for edge detection are the Sobel Operator, Scharr Operator, Roberts Cross Operator and the Prewitt Operator. Among these, Sobel is the most commonly used because of its relative immunity to noise compared to the other operators. The Sobel kernel consists of two matrices which are convolved with the image data in the x and y directions.

$$\mathbf{G}_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} * \mathbf{A} \quad \text{and} \quad \mathbf{G}_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * \mathbf{A}$$

Figure 3 Sobel Kernel [9]

The absolute magnitude of the convolution is then taken and the two values are summed. A high value indicates an edge. An appropriate threshold can be chosen to determine the validity of edge. This is necessary because certain factors can affect the perception of edges including focal blur, illumination etc. [10].

1.3.2 Frame Differencing

Frame differencing is one of the most commonly used techniques in image processing for motion estimation used in many video codecs e.g. H.264 [11]. The idea is simple; take the previous frame and subtract the current frame from it. Only the difference is then transmitted and the image is built at the decoder by summing successive frames.

1.3.3 Edge Detection and Frame Differencing

A combination of edge detection and frame differencing can also be used for motion estimation. Edge detection is first applied to get a one bit per pixel edge map of the frame. This edge map is then subtracted from a stored edge map of the previous frame. The results of ED and FD are presented in [7] which shows its resilience to data rate reduction compared to ED, FD and BS. Figure 5 shows a comparison of Information Delivery for ED+FD, ED, FD and BS against data rate.

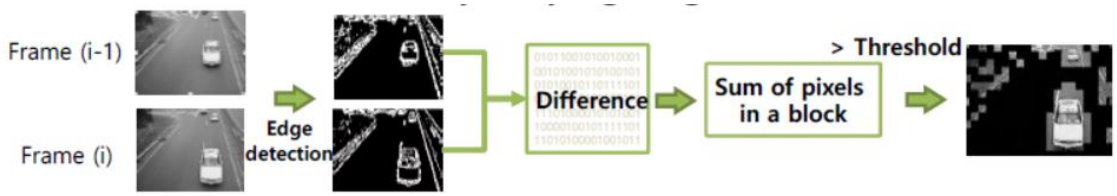


Figure 4 ED and FD Flow Diagram [7]

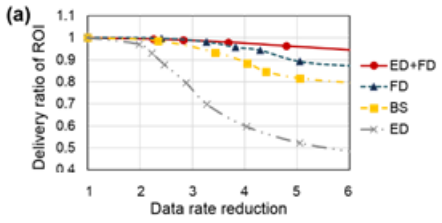


Figure 5 Comparison of various preprocessing techniques [7]

1.3.4 Image Compression

Encoding techniques are applied to the image data to reduce the overall amount of data with the end result that the number of transmissions are reduced and the energy expended at the transmitter is reduced. A number of encoding techniques exist including Huffman encoding, Run-Length Encoding, Arithmetic encoding.

1.4 The JPEG Standard

The JPEG file standard is one of the most commonly used image compression standards for digital images. JPEG consists of three main steps i.e. transform to frequency domain, quantization and finally encoding.

1.4.1 DCT

The discrete cosine transform is applied to an 8x8 block successively. The DCT gives us the spectral information within an image. This is advantageous to us because it has been observed that the bulk of the image content is concentrated around the lower frequencies and the higher frequencies make negligible to no contribution.

A two dimensional DCT is performed on the image block by first performing DCT on the rows and then performing DCT on the columns of the result (or vice-versa).

$$\begin{aligned} X_{k_1,k_2} &= \sum_{n_1=0}^{N_1-1}\left(\sum_{n_2=0}^{N_2-1} x_{n_1,n_2}\cos\left[\frac{\pi}{N_2}\left(n_2+\frac{1}{2}\right)k_2\right]\right)\cos\left[\frac{\pi}{N_1}\left(n_1+\frac{1}{2}\right)k_1\right] \\ &= \sum_{n_1=0}^{N_1-1}\sum_{n_2=0}^{N_2-1} x_{n_1,n_2}\cos\left[\frac{\pi}{N_1}\left(n_1+\frac{1}{2}\right)k_1\right]\cos\left[\frac{\pi}{N_2}\left(n_2+\frac{1}{2}\right)k_2\right]. \end{aligned}$$

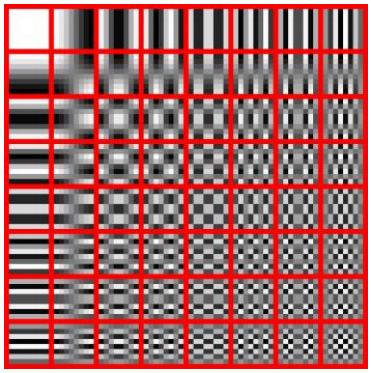


Figure 6 DCT 8x8 coefficients [12]

The first coefficient is known as the DC coefficient as it is the average of all the pixel values in the 8 x 8 block. The other 63 coefficients are the AC coefficients and represent the change within the block.

1.4.2 Quantization

Quantization is a process by which the less important coefficients can be dropped. In JPEG, the image quality can be adjusted by the quality factor. A quality factor of 100% means that no quantization is applied to the DCT coefficients. A decreasing quality factor leads to more AC coefficients being dropped.

A quantization kernel takes the form of an 8 x 8 matrix and each value is obtained by dividing the corresponding DCT coefficient with the quantization coefficient and then rounding off the result. A higher quantization coefficient results in a greater likelihood that the result will be zero.

$$Q = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}.$$

$$B_{j,k} = \text{round}\left(\frac{G_{j,k}}{Q_{j,k}}\right) \text{ for } j = 0, 1, 2, \ldots, 7; k = 0, 1, 2, \ldots, 7$$

Figure 7 Quantization formula and a typical quantization kernel [12]

1.4.3 Encoding

Next, encoding is done to reduce the overall size of all the DCT coefficients. Encoding in JPEG is a combination of two techniques i.e. Run Length Encoding and Huffman Encoding.

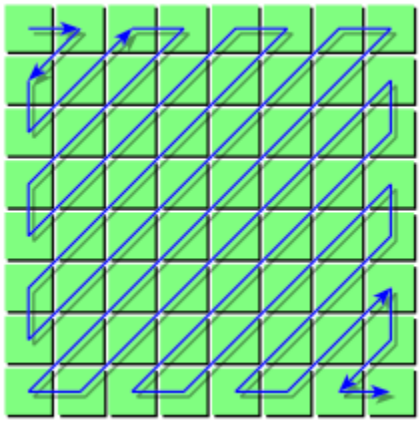


Figure 8 Ordering of DCT coefficients for encoding

First, the components of the 8 x 8 block are ordered based on their priority. Next, run length encoding is applied to these coefficients and the zero coefficients are run length encoded. The remaining coefficients are then encoded in the following manner.

1. Two symbols are created; symbol1(RUNLENGTH, SIZE) and symbol2(AMPLITUDE)
2. RUNLENGTH is the number of zeros before a nonzero coefficient represented using a 4 bit value.
3. If there are more than 15 zeros then a special symbol (15, 0), (0) is created.
4. SIZE is the number of bits required to represent the amplitude of the coefficient. It is obtained by taking the base 2 logarithm. This is also a 4 bit symbol.
5. AMPLITUDE is the amplitude of the coefficient represented in SIZE number of bits.

So each non-zero coefficient consists of an 8 bit symbol and a variable bit symbol representing its amplitude. Huffman encoding is then applied to symbol1 and symbol2 is appended to it.

Huffman encoding is a variable length encoding scheme. It builds a dictionary of the probabilities of occurring symbols within a bitstream. Using these probabilities, it assigns codes to the symbols. A frequently occurring symbol is then assigned fewer bits for its code while an infrequent symbol is assigned a higher bit code.

An ideal Huffman implementation will build a dictionary of codes by examining them and their frequencies first but we don't have that luxury in hardware as it will increase the latency manyfold. Instead a table is creating beforehand using known values. Encoding is made easier by the fact that only 160 huffman codes are needed. This is because we are only encoding symbol 1 which can take on 16 * 10 values (size of each DCT coefficient is 10 bits).

CHAPTER 2

OVERALL SYSTEM ARCHITECTURE

The overall system architecture is presented in figure 9. It is a block based pipelined architecture with each stage working on an 8x8 block of image data. The system is comprised of the following parts

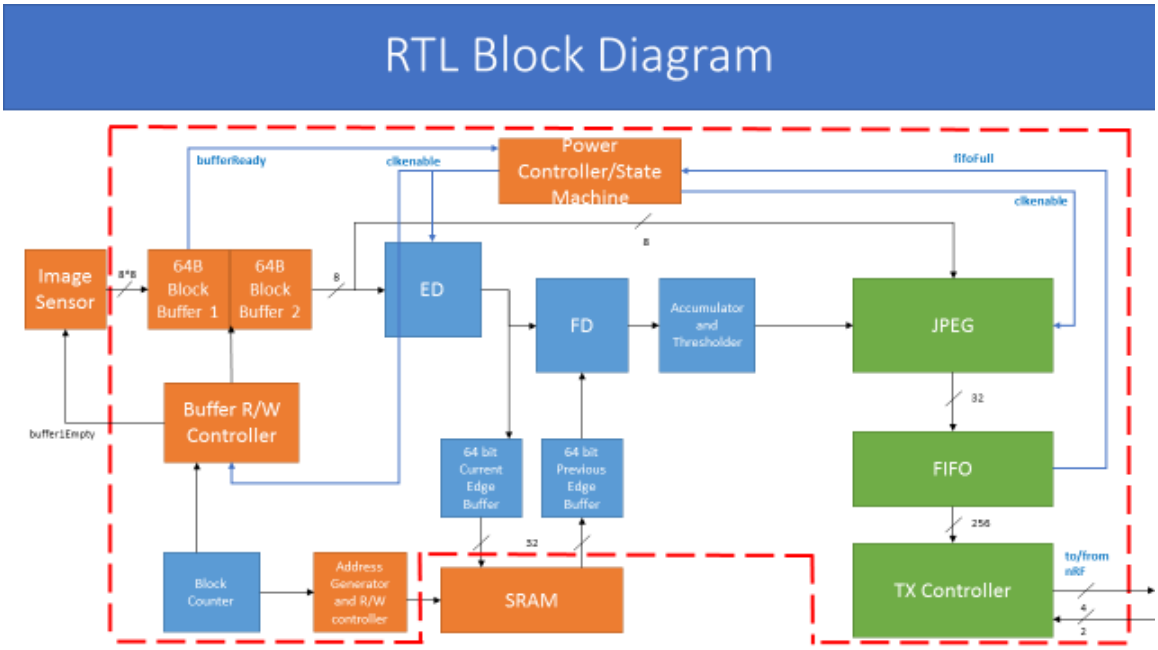


Figure 9 System Block Diagram

1. 2 64 byte Block Buffers for storing an 8x8 block of image data.
2. Preprocessor
    a. Edge Detector
    b. Frame Differencer
    c. Previous Frame Edge Buffer (64 bits)
    d. Current Frame Edge Buffer (64 bits)
    e. Accumulator and Thresholder
3. SRAM
4. JPEG Encoder

5. TX FIFO
6. TX Controller
7. System Controller

A detailed description of each component is provided in the subsequent sections. The data is manipulated in the following sequence in the pipeline

1. An 8x8 block of data is pushed onto a 64 byte block buffer 8 pixels at a time (one row)
2. The edge map of the corresponding block from the previous frame is loaded onto the previous frame edge buffer.
3. The edge detector reads pixels from the block buffer and computes the one bit edge value.
4. These one bit values are pushed on to the current frame edge buffer which are then stored in the SRAM
5. The frame differencing unit takes the current edge value and the previous edge value and performs XOR.
6. The result is passed to the accumulator which sums it up until 64 pixels have been computed.
7. Based on the result of the accumulator, the block is then read sequentially by the JPEG module for encoding.
8. While the encoder is encoding, the preprocessor is free to process another block.
9. The encoder loads the bitstream onto a 256 bit 2-level FIFO.
10. The TX controller fetches from the FIFO asynchronously and pushes it to the transmitter.
11. At any time if the TX FIFO becomes full, the rest of the system is clock gated so no data is lost.

2.1 Advantages and Disadvantages of Block Based Architecture

A block based pipelined architecture was chosen for the following reasons

1. It is resolution scalable i.e. the pipeline is not affected by the image size. It could have an impact on the frame speed (A larger frame will take more time)
2. Encoding, preprocessing and transmission can work independently.
3. For preprocessing, we don't have to wait for an entire row to do edge detection.

However there are some disadvantages as well to the block based architecture.

1. For edge detection, false edges are detected at the corner. Since we don't know in advance the pixel values in the next block, the only solution is to zero pad the edges of the kernel or replicate values at the boundary. Both solutions may mean that some edges at the boundaries of the block may be missed. A graphical depiction of this is given in figure 10.

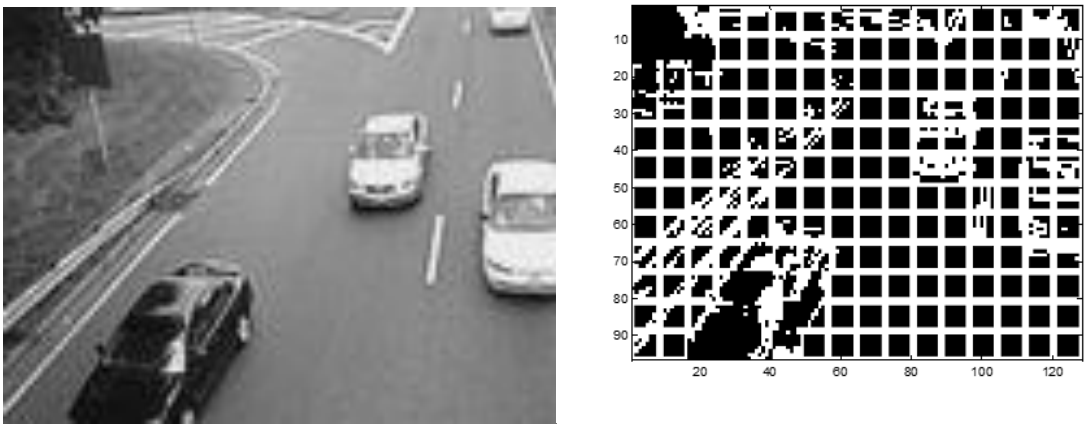


Figure 10 Edge Detected Frame

2.2 Block Buffers

Two block buffers configured in a FIFO state work as a bridge between the preprocessor and the image sensor. The image sensor pushes data, on block buffer 1, 8 pixels at a time (one row of a block). If block buffer 2 is not being used by the preprocessor or the encoder,

the data from block buffer 1 is pushed on to block buffer 2. A block diagram of the buffers is given in figure 11.

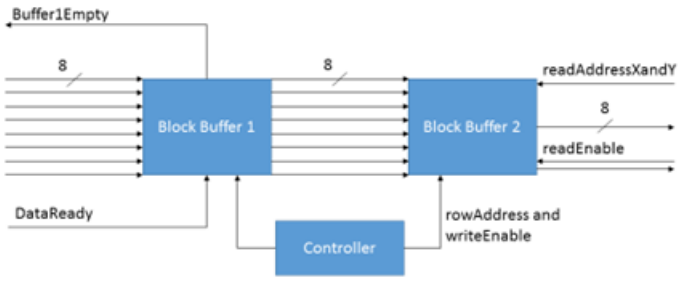


Figure 11 Block Buffer Block Diagram

Once the contents of buffer 1 are copied, a signal buffer1Empty is asserted to let the image sensor know that buffer 1 is ready to be written. Buffer 1 is row addressable for both reading and writing while Buffer 2 is row addressable for writing only. Reads from buffer 2 are performed one pixel at a time. A state machine controls this cycle of reads and writes from the image sensor to the buffers and the subsequent reads. The sequence of operations can be seen in figure 12.

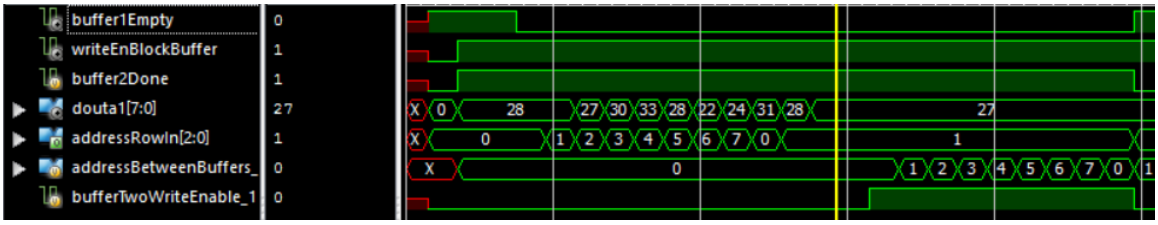


Figure 12 Block Buffer Timing Diagram

2.3 Preprocessing Module

The preprocessing module is the decision making module as it determines what blocks are to be encoded and what blocks can be dropped without losing information. It consists of the following modules

1. Edge Detector
2. Frame Differencing unit

3. Previous Frame Edge Buffer (64 bits)
4. Current Frame Edge Buffer (64 bits)
5. Accumulator and Thresholder

2.3.1 Edge Detector

In the edge detector we need to perform convolution between the block and the sobel kernel. Each pixel's edge value is calculated by performed by loading 9 pixels from the block buffer and then multiplying by the flipped sobel kernel and adding the result. The next pixel is computed by shifting the image data in the left direction and performing the same operation.

It can be immediately observed that these loads are redundant and are wasting precious cycles. Thus, after the first load for each row, only the next column is loaded thus saving 6 loads * 7 pixels = 42 load cycles per row. Figure 13 demonstrates how these loads are reduced by reusing already loaded values.

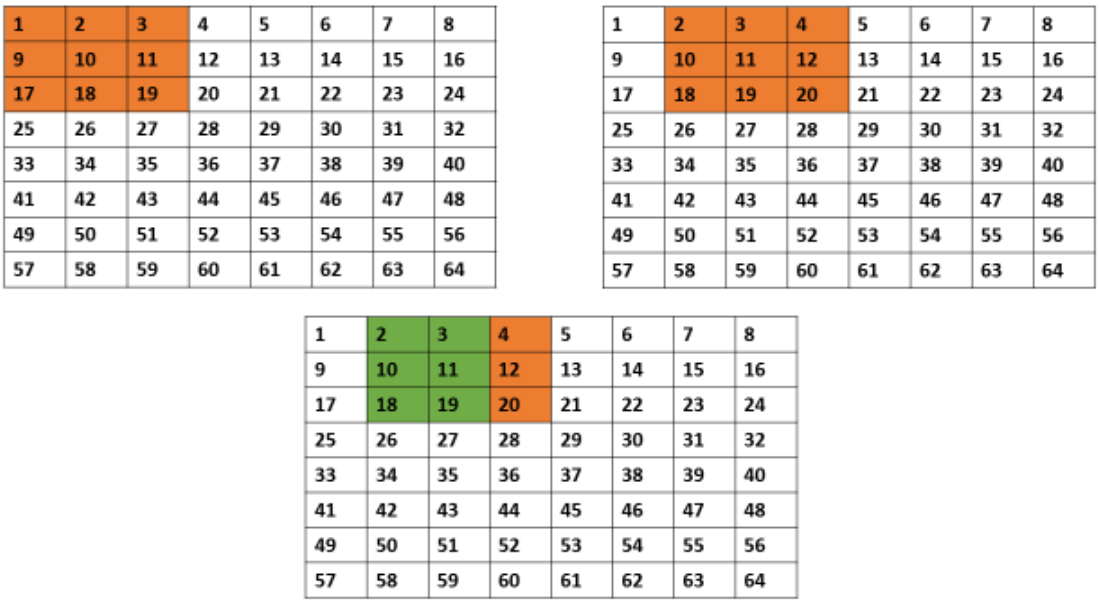


Figure 13 Reusing values to reduce number of loading operations

2.3.2 Frame Differencing Unit

The frame differencing unit consists of a simple XOR gate which takes inputs from the Edge Detector and the Previous Frame Edge Buffer.

2.3.3 Current Edge Frame Buffer

The current edge frame buffer stores the output of the edge detector. After 64 bits are processed and stored within the buffer, the buffer controller writes the contents of this buffer to the SRAM 32 bits per clock cycle i.e. 2 cycles of writing.

2.3.4 Previous Edge Frame Buffer

The previous edge frame buffer stores the corresponding block of the previous frame edge map. At the start of a computation cycle, the buffer controller reads the contents of the SRAM and writes to this buffer 32 bits per clock cycle i.e. 2 cycles of SRAM reading. Once this is done, the contents of the buffer are output one bit at a time to the frame differencing unit.

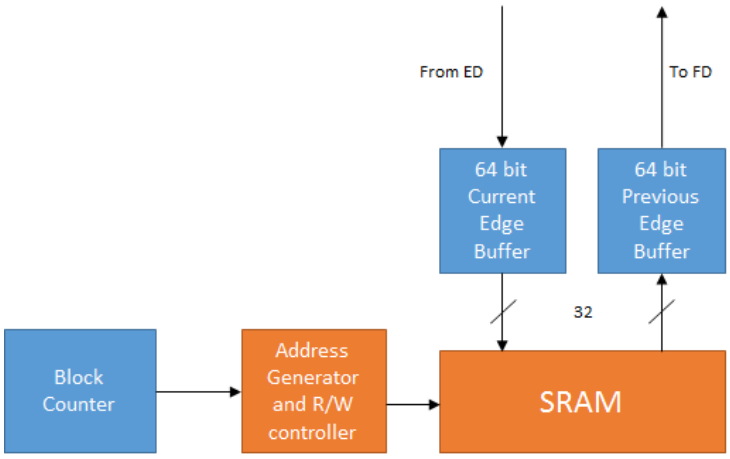


Figure 14 Edge Map Buffers Block Diagram

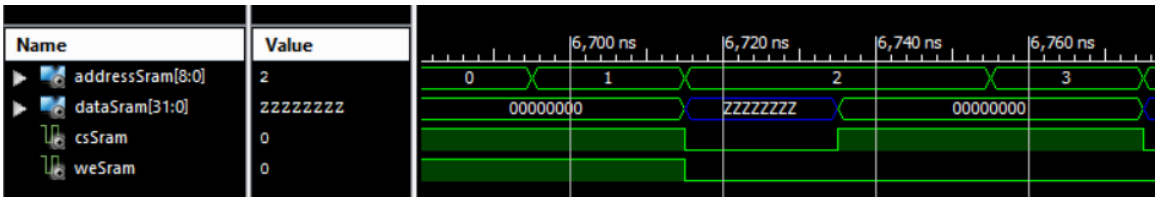


Figure 15 Edge Map Buffers Timing Diagram

2.3.5 JPEG

An open source JPEG core was used courtesy of David Lundgren from opencores.org. The provided core is capable of full JPEG encoding for three color channels i.e. YCbCr. We are only using the luminance (Y) channel core as our input data is composed of grayscale images.

The JPEG module is composed of three main modules corresponding to the three operations performed in JPEG i.e. DCT, quantization and Encoding. The block data is input serially to the encoder in 64 cycles. The core outputs the bitstream in the form of 32 bit packets. DCT takes up the largest area of all the modules.

2.3.6 TX FIFO

TX FIFO is needed because of two main reasons

1. Different size payload
2. nRF operates at a different clock frequency

The JPEG core output packet is composed of 32 bits at any instant while the maximum payload size of the nRF is 256 bits. An asynchronous FIFO two level FIFO of 256 bit width acts as a buffer between the transmission controller and the JPEG.

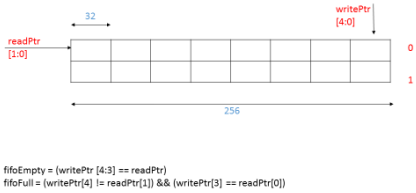
The FIFO is designed so that asynchronous reads and writes are possible. Two pointers are maintained, a read pointer and a write pointer. With reference to the figure 16(a), there are only two read locations so the read address is only one bit. To make it a circular FIFO an additional bit is added giving us a 2 bit read pointer. By the same logic, the write pointer is (1+4) = 5 bits.

In addition to the pointers, two signals are generated which indicate an empty FIFO and a full FIFO. A FIFO empty is needed for the read side of the FIFO while the FIFO full signal is needed for the write side of the FIFO. These signals are determined by the following conditions.

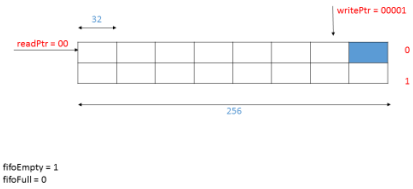
fifoEmpty = (writePtr [4:3] == readPtr)

fifoFull = (writePtr[4] != readPtr[1]) && (writePtr[3] == readPtr[0])

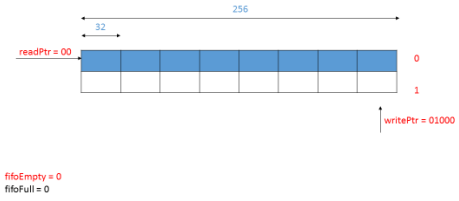
A FIFO full condition occurs when both pointers are pointing to the same location but the circular bit is reversed i.e. all possible locations have been written to. A graphical depiction is given in the figure 16.



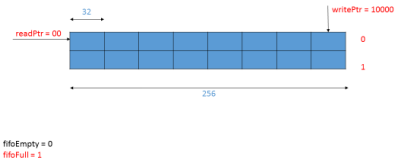
(a)



(b)



(c)



(d)

(e) (f)

Figure 16 (a) FIFO Structure (b) FIFO writing (c) One payload written to FIFO (d) Full FIFO condition (e) FIFO reading (f) FIFO empty after reading

2.3.7 Transmission Controller

For wireless transmission a Nordic NRF2L01+ is being used. It is a 2.4 GHz transceiver chip that can provide an air data rate up to 2 Mbps. The transceiver is configured using a SPI interface. At power up, the chip is configured by writing to its CONFIG register and a power up time of 1.5 ms is provided.

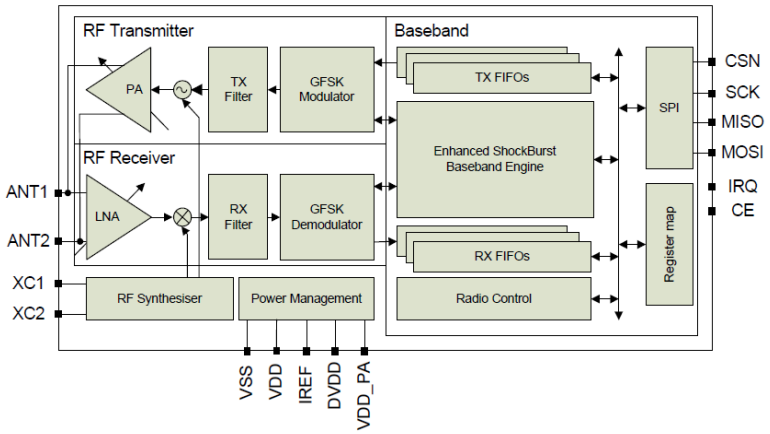


Figure 17 nRF module Block Diagram

Once this is done the chip is ready for transmission and reception. The nRF module supports a transmission size of 1-32 bytes at a time. The state diagram for the TX controller which interfaces with the nRF is given in figure 18. At startup, the system enters the CONFIG state where it supplies the configuration commands to the nRF.

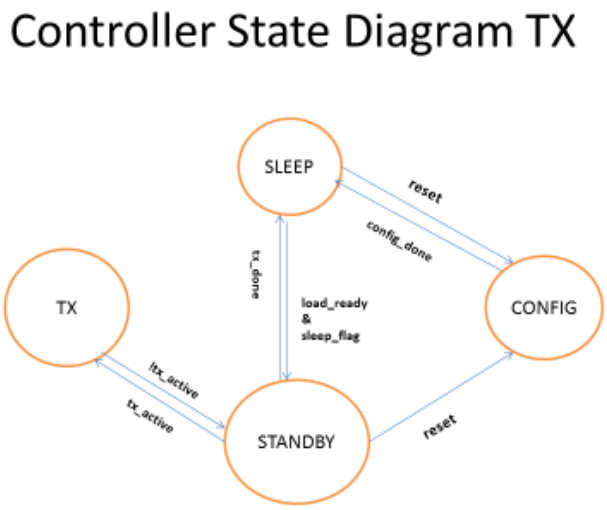


Figure 18 TX Controller state diagram

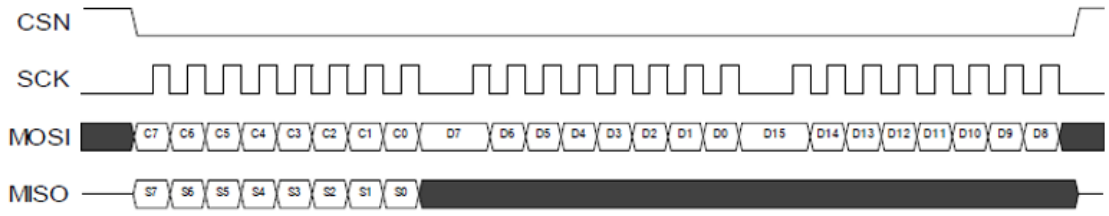


Figure 19 Write Operation to nRF module timing diagram

After that it enters the SLEEP state and stays there until it has a packet to transmit. When the transmit buffer is ready, the system switches to the STANDBY state. During the STANDBY state, the TX controller pushes the payload onto the nRF. A timing diagram of this operation is given in figure 20. After pushing the payload the TX controllers enters the TX state where it waits for the nRF module to send the payload. An interrupt on the IRQ pin appears when the payload is transmitted over the air.

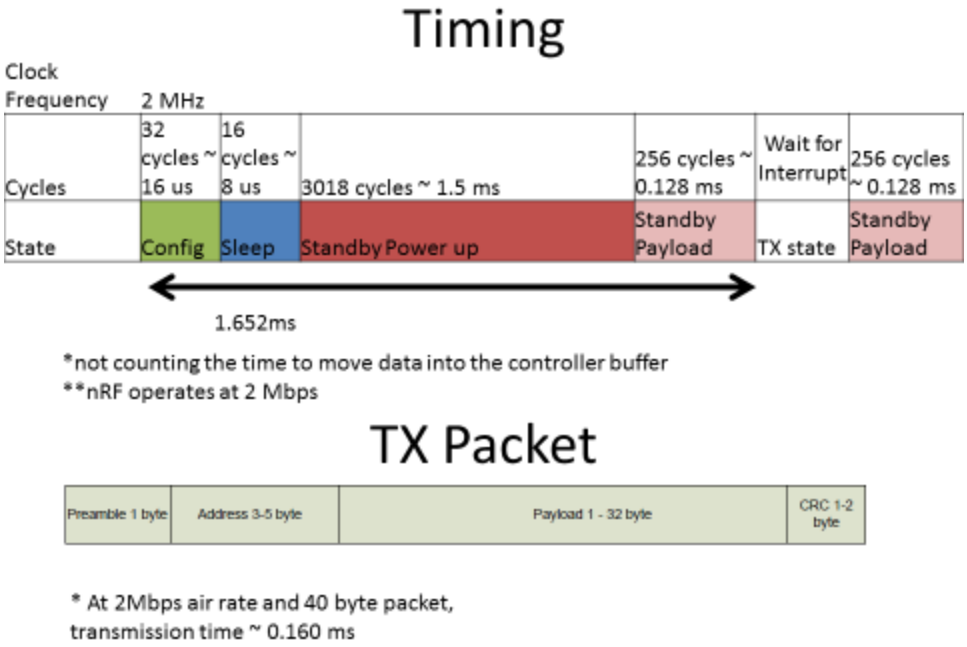


Figure 20 Sequence of operations and timing diagram for TX Controller

2.4 Adaptive Preprocessing

Depending on channel conditions, it is desired that the image processor adjusts the content delivery accordingly. For example, a harsh channel condition would impose a stricter edge and block threshold and reduce the QF of the JPEG module.

The system is configured with a synchronous interrupt. At each interrupt, the threshold registers are reconfigured to a user provided value. The value is provided through an external interrupt register. A system controller keeps track of channel conditions and controls it accordingly.

2.5 Clock gating

The transmitter in our implementation runs at 2 MHz while the system is designed to run at a higher clock frequency. This means that more often than not, the TX FIFO will be full

while transmission is occurring. In this case, the preprocessor and encoder need to be stopped so that loss of data does not occur.

2.5.1 Full TX FIFO Clock Gating

A simple way to preserve the state of the system is to clock gate it. Whenever a FIFO full condition occurs, the clock signal to the Preprocessor and encoder is de-asserted so that no switching takes place in the registers. During this phase, the only contribution to computation power is the leakage power.

2.5.2 Block Level Clock Gating

Another opportunity for clock gating arises when we have the case where the preprocessor is done with a block before the image sensor provides it with a new block. Similar to above, we are wasting clock cycle while not doing any work so we can safely clock gate the preprocessor.

2.6 Automatic encoding for empty blocks

During system operation, a number of blocks will be dropped by the preprocessor. It is not desirable that they be encoded again by the encoder. We can take advantage of the fact that a symbol for an empty block is pre-determined in JPEG. A signal is asserted by the preprocessor when it drops a block. The encoder then inserts the empty block symbol in the bitstream. A lot of power is saved in this way because the majority of the power consumption in the JPEG module comes from the DCT computations.

CHAPTER 3

RESULTS AND CONCLUSIONS

3.1 Test Setup

To test the output of the image processor, the FPGA was connected to the computer using a serial port to verify its output bitstream. An nRF module was connected to the output headers of the FPGA evaluation board to perform transmission of data packets. An nRF transceiver configured in receiver mode was used to verify reception of packets.

The serial port extracted data from the TX controller and transmitted to the computer at a rate of 9600 baud. Figure 21 shows the test setup. A program SerialWatcher™ was used to receive and verify the output of the TX controller.

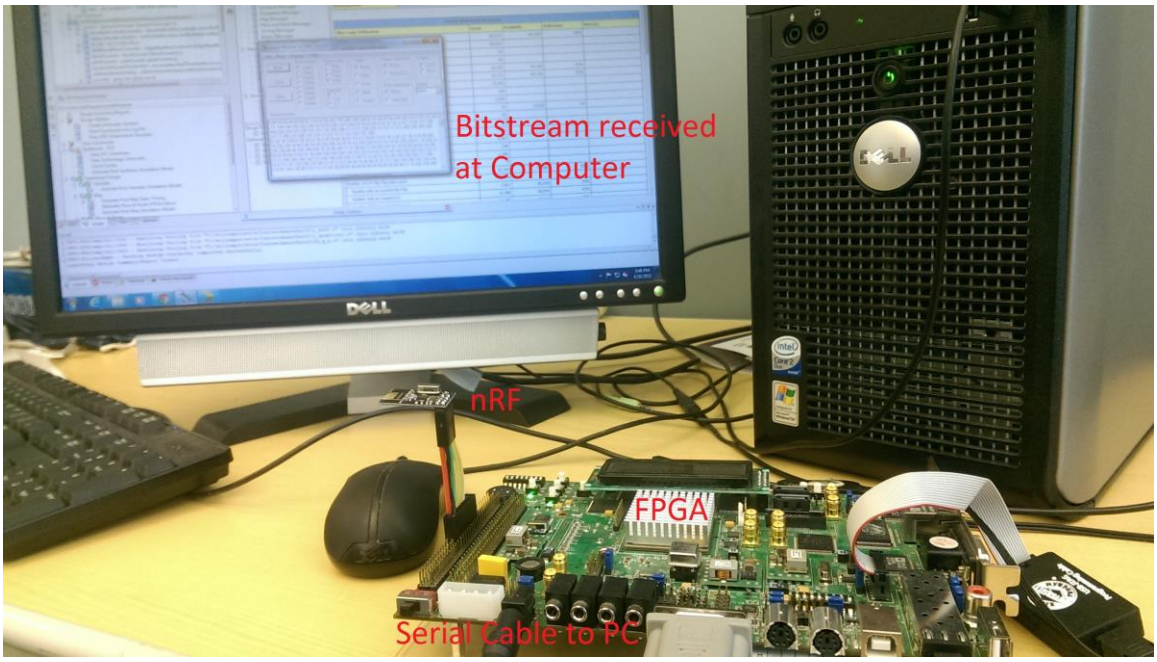


Figure 21 Test Setup for Verification of RTL

Figure 22 shows the two sample frames, from our traffic camera data set, which were loaded on to the FPGA ROM to be used by the processor.

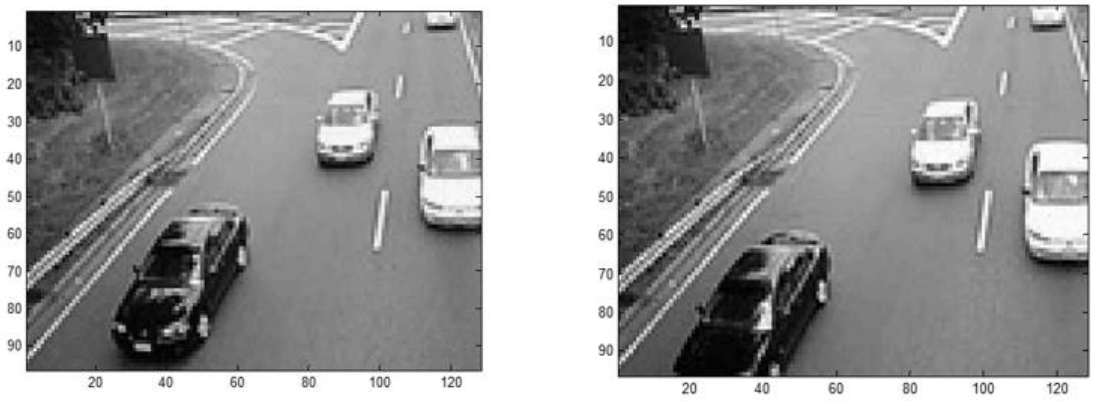


Figure 22 Test Frames 1 and 2

The frame results for a few edge thresholds and block thresholds are given in the figure 23 and 24. The block threshold is a crucial factor in determining whether a block should be kept or not. More blocks are dropped when it is increased.

Edge threshold determines which gradients qualify as edges. This threshold is heavily dependent on the illumination of the scene and has to be adjusted accordingly.

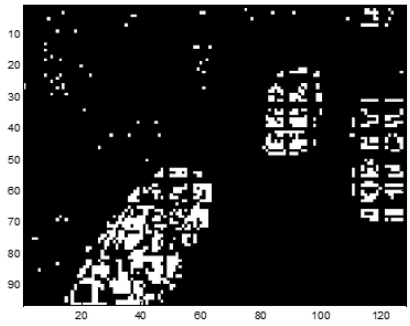
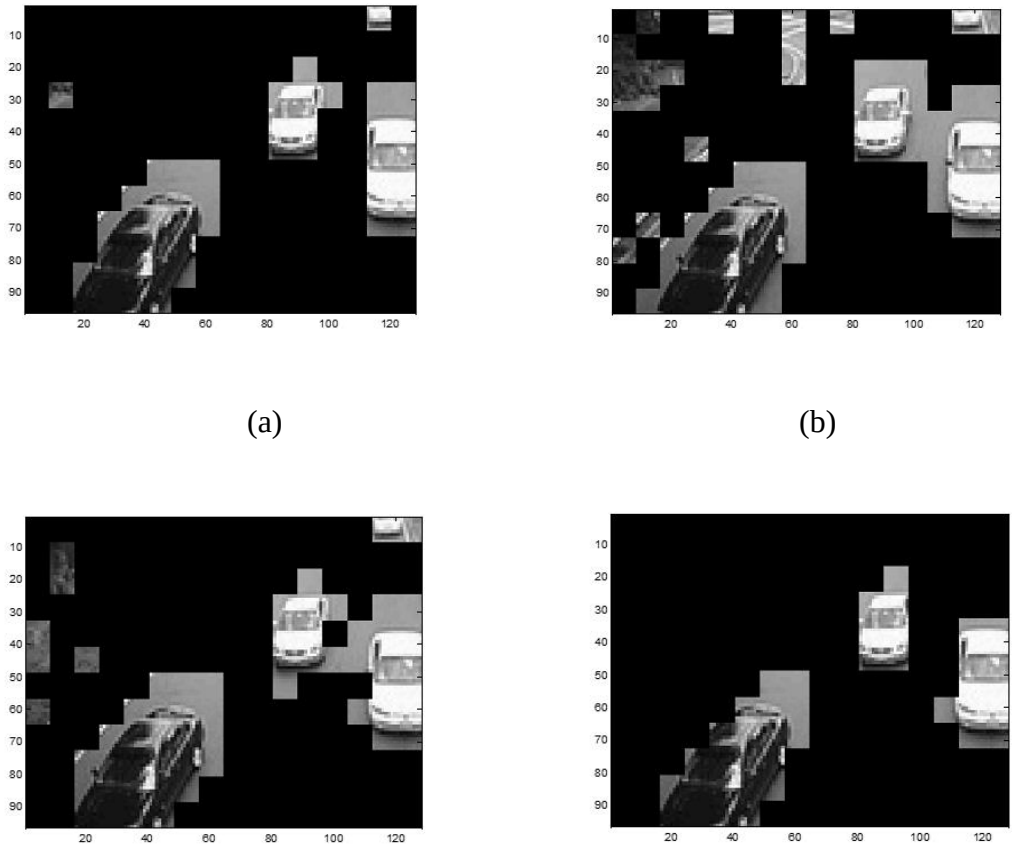


Figure 23 Edge Map after Frame Differencing



(a) (b)

(c) (d)

Figure 24 Frame 2 Results (a) Edge Threshold = 200, Block Threshold = 5 (b) Edge Threshold = 200, Block Threshold = 2 (c) Edge Threshold = 100, Block Threshold = 5 (d) Edge Threshold = 100, Block Threshold = 10

The volume of data generated from the preprocessor for the above cases is presented in the table 1. It can be seen that the file can be further compressed with preprocessing while maintaining a respectable information delivery ratio. The information delivery ratio is calculated here by counting the number of blocks with moving objects i.e. cars in the frame. Note that not all the blocks that are sent contain information about moving objects. All results are for JPEG QF of 50 percent.

Table 1 Comparison of volume of data generated

Test	File Size (B)	File Size After JPEG (B)	Information Delivery (%)	Blocks Sent	Compression Ratio After Encoding
No Preprocessing	12288	1724	100	192	100.0
Edge Threshold = 200, Sum Threshold = 2	4736	1040	100	74	60.3
Edge Threshold = 200, Sum Threshold = 5	3008	752	84.9	47	43.6
Edge Threshold = 100, Sum Threshold = 5	3840	872	92.5	60	50.6
Edge Threshold = 100, Sum Threshold = 10	2560	684	71.70	40	39.7

Table 2 details the power consumption calculated using current values from the nRF datasheet and with an operating voltage of 3.3 V. The transmission time per block was calculated from the point where the data is completely loaded to the nRF to the time the interrupt for data sent is received at the IRQ pin from the nRF.

Table 2 Electrical and Timing Characteristics of nRF[1]

Operating Voltage of nRF	3.3 V
Transmission time per Block	0.023 ms*
Current Consumption during Transmission	11.3 mA
Loading time per Block	0.128 ms
Current Consumption during Loading	285 uA
Power Consumption during Loading	37.3 mW
Power Consumption during TX	0.94 mW

[1] Current values taken from datasheet, *Measured from FPGA

The nRF power consumption is detailed in table 2. As expected it has a higher power consumption during transmission then when data is being loaded onto it. Table 3 gives an idea of the number of resources used by each block of the processor. The whole system (given by the row System in table 3) is composed of 13795 FFs and 16541 LUTs. Of This the JPEG encoder takes up about 11588 FFs and 15390 LUTs i.e. about 84 % of the logic area. The same is reflected in the power distribution as the JPEG module consumes 72 % of the total computation power[2]. All power values are for a 50 MHz clock.

Table 3 System Usage and Area Statistics

Name	# FFs	# LUTs	# SRLs	# BRAMs	# DSPs	# CARRY4s
⊟ Hierarchy total	32179	23153	668	31	64	2775
⊟ testPreprocessorWithSerial	11 / 32179	142 / 23153	0 / 668	0 / 31	0 / 64	0 / 2775
⊟ System	22 / 13795	29 / 16451	1 / 22	0 / 5	0 / 64	0 / 2658
⊞ ED_FD	0 / 99	0 / 267	0	0	0	0 / 28
addressMuxBlockBuffer	0	8	0	0	0	0
addressGeneratorJpeg	6	7	0	0	0	0
accumulatorandThresholder	7	9	0	0	0	0
⊞ SRAMandMemController	5 / 166	6 / 107	0	0	0	0
blockCounter	8	10	0	0	0	2
⊞ blockBuffersAndController	18 / 1116	16 / 332	0	0	0	0
txFIFO	776	283	0	0	0	0
pixelCounter	7	9	0	0	0	0
⊞ encoder	69 / 11588	287 / 15390	0 / 21	0 / 5	0 / 64	0 / 2628

Table 4 Power Distribution in Processor

POWER CONSUMPTION (W)	
Preprocessor	3.76 mW
JPEG	13.01 mW
Total Power	18.96 mW

The utility of this system can be judged by the amount of power savings it generates. Power savings as a result of encoding are presented in table 5. It can be seen that the power consumption drops by one order of magnitude simply due to encoding. Energy values presented here are for one base frame that is completely transmitted (i.e. no preprocessing and no dropped blocks). Energy values are computed using power values from table 4 and

[2] Power values generated from Xilinx Xpower Analyzer for FPGA

timing values for encoding which is 5 us per block for a total of 192 blocks. Figure 25 gives a visual depiction of the energy savings.[3]

Table 5 Power Savings after Encoding

Test	Bytes	Payloads (32 byte)	Transmission Energy	Computation Energy	Total Energy
No Encoding	12288	384	377 uJ	0	377 uJ
Encoding	1724	54	52.9 uJ	12.4 uJ	65.4 uJ

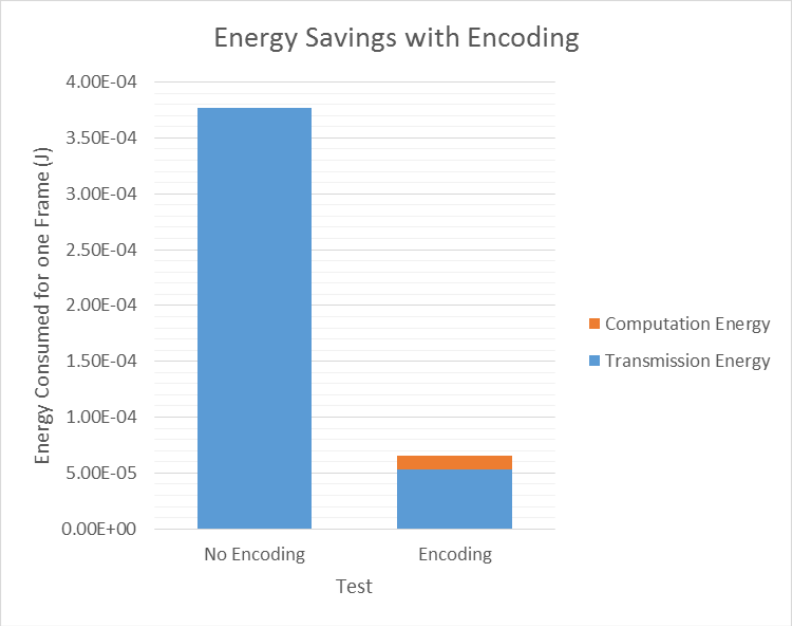


Figure 25 Energy Savings with encoding

Building up on that, the addition of the preprocessor should grant us additional power savings. The energy values in table 6 show that once again, preprocessing lowers the energy consumed per frame by another order of magnitude. However, we should make note

[3] TX power values approximated from time of transmission per payload and data sheet power values

of the fact that this number is dependent on the number of moving objects within the frame. A high activity within the frame will lead to a higher transmission volume increasing the energy consumed per frame.

Table 6 Energy Consumption Comparison with Preprocessing for one Frame

Test	Bytes	Payloads (32 byte)	Transmission Energy	Computation Energy	Total Energy
No Preprocessing	1724	54	52.9 uJ	12.5 uJ	65.4 uJ
Edge Threshold = 200, Sum Threshold = 2	1040	33	31.9 uJ	8.24 uJ	40.2 uJ
Edge Threshold = 200, Sum Threshold = 5	752	24	23.1 uJ	6.48 uJ	29.6 uJ
Edge Threshold = 100, Sum Threshold = 5	872	27	26.8 uJ	7.33 uJ	34.1 uJ
Edge Threshold = 100, Sum Threshold = 10	684	21	21.0 uJ	6.03 uJ	27.0 uJ

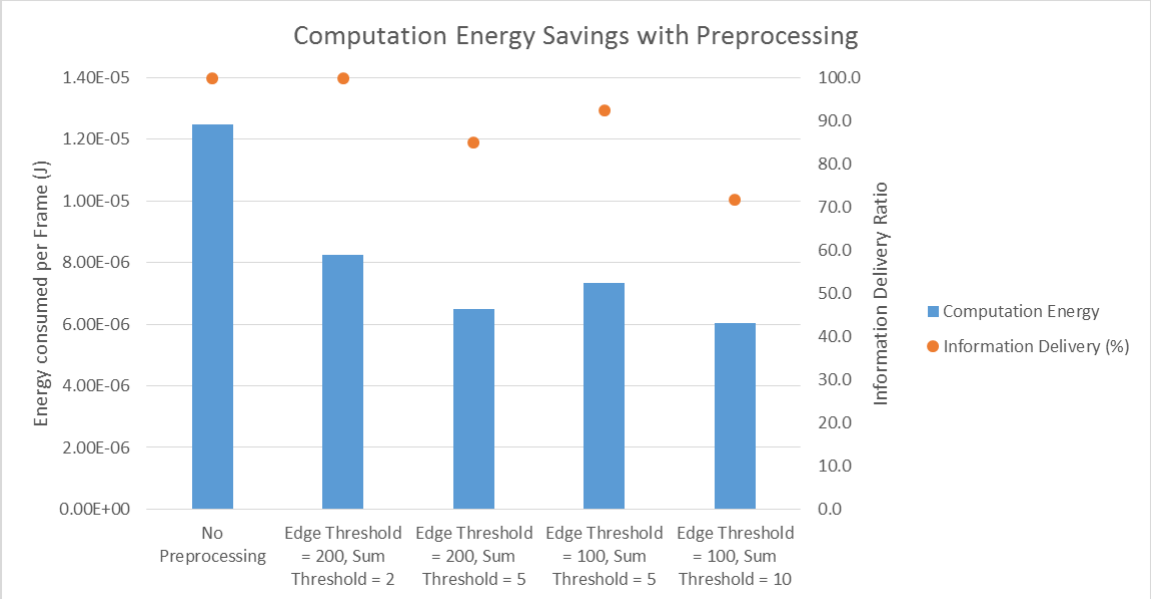


Figure 26 Computation energy savings with preprocessing

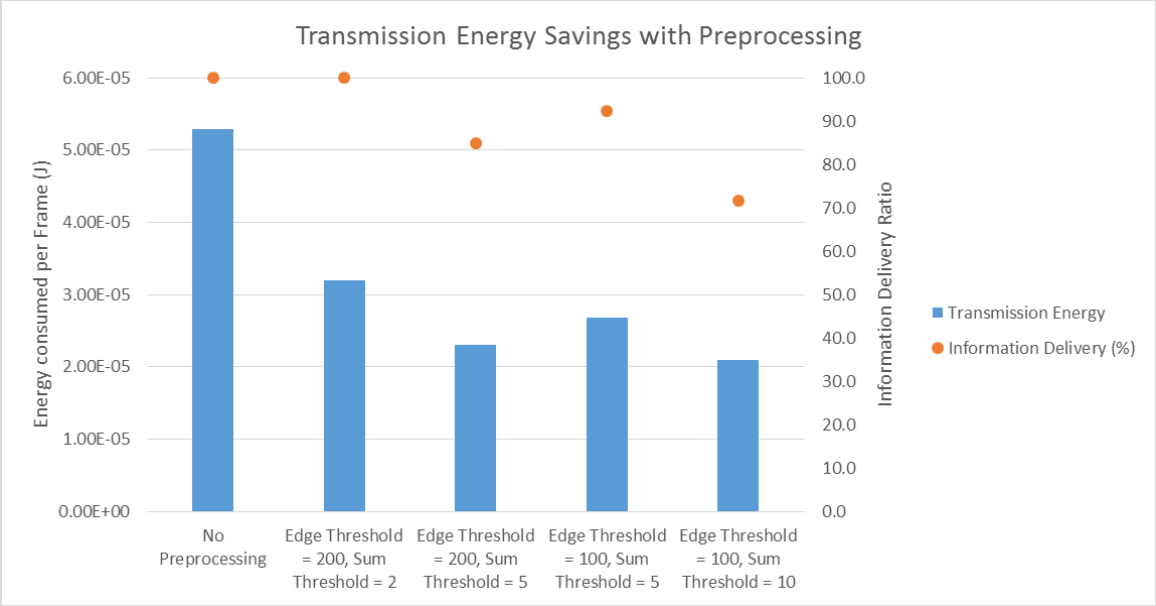


Figure 27 Transmission energy savings with preprocessing

3.2 Conclusions and Future recommendations

In this work, a complete content aware system was implemented and demonstrated on a FPGA including wireless transmission. Through this scheme we are able to achieve a 54.7 % reduction in the total energy used for wireless transmission of a single frame while maintaining the information delivery ratio above 85 %.

The next step would be to interface this system with a commercial imaging unit and extract a video from the output of the system. Another direction would be to implement an automated controller that reconfigures the thresholds by sensing channel conditions. A statistical analysis is also needed to verify the power reduction trend observed for this set of images.

REFERENCES

[1] Nick Kanopoulos, Nagesh Vasanthavada, and Robert L. Baker, "Design of an Image Edge Detection Filter Using the Sobel Operator," *IEEE Journal of Solid State Circuits*, vol. 23, no. 2, pp. 358-367, April 1988.

[2] Gyuoho Kim, "A Millimeter-Scale Wireless Imaging System with Continuous Motion Detection and Energy Harvesting," in *Symposium on VLSI Circuits Digest of Technical Papers* , Honululu, 2014, pp. 1-2.

[3] Jaehyuk Choi, Seokjun Park, Jihyun Cho, and Euisik Yoon, "A 3.4 uW Object-Adaptive CMOS Image Sensor With Embedded Feature Extraction Algorithm for Motion-Triggered Object-of-Interest Imaging," *IEEE JOURNAL OF SOLID-STATE CIRCUITS*, vol. 49, no. 1, pp. 289-300, 2014.

[4] A. Chefi, A. Soudani, and G. Sicard, "A CMOS image sensor with low-complexity video compression for wireless sensor networks," in *New Circuits and Systems Conference (NEWCAS), 2013 IEEE 11th International*, Paris, 2013.

[5] Shoushun Chen, Wei Tang, Xiangyu Zhang, and E. Culurciello, "A 64 times 64 Pixels UWB Wireless Temporal-Difference Digital Image Sensor," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 20, no. 12, pp. 2232 - 2240, 2012.

[6] Kerem Irgana, Cem Ünsalanb, and Sebnem Bayderea, "Low-cost prioritization of image blocks in wireless sensor networks for border surveillance," *Journal of Network and Computer Applications*, vol. 38, pp. 54–64, 2014.

[7] Jong Hwan Ko, Burhan Ahmad Mudassar, and Saibal Mukhopadhyay, "An Energy-Efficient Wireless Video Sensor Node with Content-Aware Pre-processing for Moving Object Surveillance," *Embedded Systems Letters*, 2015.

[8] Jon Mcloone. Wikipedia. [Online]. http://en.wikipedia.org/wiki/File:EdgeDetectionMathematica.png#/media/File:EdgeDetectionMathematica.png

[9] Edge Detection - Wikipedia. [Online]. http://upload.wikimedia.org/math/9/8/a/98a9514f83f3e81531216e92efb212a3.png

[10] Wikipedia. [Online]. http://en.wikipedia.org/wiki/Edge_detection

[11] H.264/MPEG-4 AVC. [Online]. http://en.wikipedia.org/wiki/H.264/MPEG-4_AVC

[12] Wikipedia. [Online]. http://en.wikipedia.org/wiki/JPEG